A QoS-Aware Middleware for Dynamic and Adaptive Service Execution.

Chen Wang

A QoS-Aware Middleware for Dynamic and Adaptive Service Execution*

Chen Wang

INRIA/IRISA, F-35708 Rennes, France
{chen.wang}@inria.fr

Abstract. Service-Oriented Architecture (SOA) provides a flexible solution for building loosely coupled distributed applications. Complex applications can be designed by defining a business process that composes a set of independent software modules called services. In this scenario, each service can be selected and bound dynamically at run-time among a set of candidates that provide the same functionality but differs in quality of service (QoS). However, the QoS values advertised by partner services are not always ensured at run-time. In response to the dynamic execution environment, the execution of a business process has to be adapted on-the-fly in case that a global QoS violation is predicted. In this paper, we introduce a QoS-aware middleware system for dynamic and adaptive service execution. The run-time service selection is modeled as an optimization problem based on user's end-to-end QoS constraints and preferences on the service composition level. In contrast to the centralized execution engine adopted by most of traditional approaches, the execution of a service composition is decentralized in the middleware. Moreover, Program Evaluation and Review Technique (PERT) and Heartbeat Failure Detector ($\mathcal{HB}$) are introduced as effective approaches to predict global QoS violations and draw appropriate adaptation decisions.

1 Introduction

Service-Oriented Architecture (SOA) is adopted today by many businesses as an effective approach for building software applications that promotes loose coupling between software components. From the viewpoint of SOA, complex applications, often referred as business processes or service compositions, can be built by defining a workflow that composes and coordinates different services available via the network.

In the context of dynamic execution, a workflow is defined by composing a set of abstract activities as place holders. Each activity is bound to a suitable partner service, which is selected at run-time from a set of functional equivalent candidates with different non-functional properties such as quality of service (QoS). Service selection introduces an optimization problem which can be solved either

* The research leading to these results has received funding from the European Communitys Seventh Framework Programme [FP7/2007-2013] under grant agreement 215483 (S-CUBE).

locally or globally. The local approach selects the best candidate for each activity separately. Accordingly, it is more efficient but the end-to-end QoS cannot be ensured. By contrast, the global approach identifies a service composition that can meet the requester's end-to-end QoS requirements. In [1], the authors prove that global service selection is equivalent to a Multiple-choice Multi-dimensional Knapsack Problem (MMKP), which is NP-Hard. As a result, identifying the best service composition often results in a higher time complexity.

During the execution, run-time QoS is determined by the dynamic execution environment so that the expected QoS is not always ensured. In addition, the infrastructure failure can lead a service undeliverable. The adaptive execution reflects the capability to recompose a (part of) workflow on-the-fly in case that the global QoS violation is predicted. The prediction is based on monitoring events such as the detection of a delay on the execution of an activity or the crash of a partner service. The recomposition aims at identifying a new service composition by re-selecting services for the unexecuted activities, by which the requester's end-to-end QoS constraints can still be guaranteed. Otherwise, the primal goal is to minimize the negative effect caused by this QoS violation. For example, if an activity has to be re-executed, the global QoS constraints can hardly satisfied. In this case, the recomposition of services identifies a new composition with shortest execution time while all the other QoS dimensions are still satisfied.

This paper introduces a QoS-aware middleware system for dynamic and adaptive service execution. The service selection is modeled as a Mixed Integer Linear Programming (MILP) problem based on the requester's global QoS constraints (such as limited budget and time) and preferences on different QoS attributes. Compare to traditional approaches which implement a centralized execution engine, the execution of a service composition is decentralized in the middleware system, which has advantages on dealing with scalability and fault tolerance. In addition, two approaches are introduced for monitoring run-time QoS: Program Evaluation and Review Technique (PERT) chart is used to predict the violation of global execution time caused by the delay on the execution of an activity; to deal with the infrastructure failures, an overlay is implemented based on Heartbeat Failure Detector ($\mathcal{HB}$) to draw an adaptation decision as early as possible. The adaptation process is formalized as a re-composition of a part of workflow. Based on the problem complexity and the execution state, either a feasible or optimal solution is identified to meet the crucial requirement of limited execution time.

The rest of paper is organized as follows. In Section 2, the middleware architecture is presented and the system model is defined. Section 3 introduces the global service selection approach that can best meet the user's end-to-end requirements. Section 4 discusses distributed execution, run-time monitoring and adaptation. Experimental results are studied in Section 5. In Section 6, some of the main existing approaches for dynamic and adaptive service execution are introduced. Finally, conclusion and future work are addressed in Section 7.

2 Middleware Architecture and System Model

2.1 Middleware Architecture

The middleware architecture is shown in Figure 1, all components are organized into three levels. Web Service Representation level resolves the abstract representation of Web services. In the middleware system, all available functionalities are provided by a set of Abstract Services (AS), which can be either atomic or composite. An atomic AS, defined by an Abstract Web Service (AWS) component, does not require the collaborations with other ASs to provide a certain functionality. It is the middleware-level reflection of a specific concrete Web service: on one hand, it advertises functional interfaces and quality levels related to this concrete Web service; on the other hand, it outsources the calculation to this concrete service in response to service invocations. By contrast, a composite AS is represented by an Abstract Business Process (ABP) component which defines an Abstract Workflow (AWF) to achieve the ultimate business goal. Each workflow activity only specifies functional requirement at the designing phase. Suitable service providers, referred as an AS in the middleware system, are selected and bound at run-time. Thus, an ABP plays the roles of both service provider and consumer. Different from AWS, it only interacts with the components defined within the middleware system.

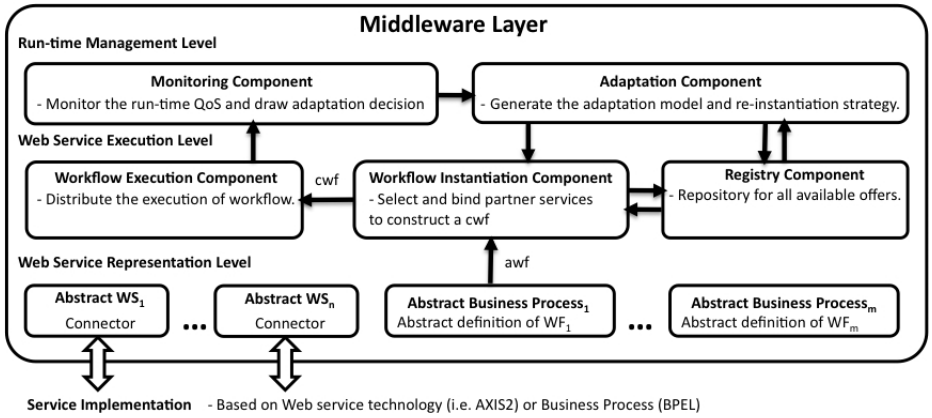


Fig. 1. Middleware Architecture

Web Service Execution level manages service executions and invocations. ASs advertise their services by defining and publishing *offer*s to the Registry component. An offer can be regarded as a contract between service requester and provider, which specifies both functional and non-functional characteristics of a service delivery. Registry acts as a directory maintaining the information of all currently available offers. Each request to an ABP is associated with a global QoS

constraint specified by the requester (i.e., the total price and execution time). In response to this request, an ABP instantiates and executes its predefined workflow. The Instantiation component is responsible for building a concrete workflow by selecting a suitable AS (can be either an AWS or an ABP) for each activity. The selection is based on all currently available offers in the Registry and the global QoS constraints specified by the requester. Once a concrete workflow is constructed, the Execution component distributes its execution by creating and forwarding a group of coordination messages to all participants.

Run-time Management level monitors and adapts service executions at run-time. Since the expected QoS values are not always guaranteed at run-time due to the dynamic and distributed execution environment, the Monitoring component monitors run-time QoS of each partner service. Once a QoS violation is predicted, it updates the global view of workflow execution state and draws adaptation decision. The Adaptation component then suspends the execution by saving the breakpoints and generates an adaptation model based on the current state of workflow execution. The adaptation is modeled as a re-instantiation process: partner services are re-selected for the unexecuted activities so as to guarantee the global QoS constraints. If a solution is found, the new concrete workflow is forwarded to the Execution component to resume the execution from the breakpoints. The dynamic and adaptive execution of a service composition follows the loop of instantiation, execution, monitoring and adaptation (if necessary). In the following sections, we are going to detail each process.

The middleware system provides a set of Web-based interfaces for service providers to create and manage abstract services. If a service is already implemented, an AWS can be created automatically by providing the WSDL file related to service implementation. On the other side, if a provider defines a service by describing an abstract workflow whereas without specifying implementation detail, he can create an ABP by uploading the workflow description file (such as BPEL file). Furthermore, service providers can manage their ASs through user interface,i.e., it is possible to define quality levels by creating, modifying or canceling his offers in the Registry component. In addition, a set of Web-tools are developed for administrators of the middleware system. With the aid of these tools, the administrators can manage and maintain the database of Registry, modify run-time policies for service instantiation and adaptation, view the system logs and so on.

2.2 Service Composition Model

An ABP defines an abstract workflow that specifies business logic and execution order. An abstract workflow is a collection of interrelated activities that function in a logical sequence to achieve the ultimate business goal, denoted as $awf = \{t_i | 1 \leq i \leq N\}$, where t_i represents the i^{th} workflow activity. A concrete workflow is constructed by binding each activity to a service provider. Most models proposed in the literature assume that a provider delivers a service only on one quality level. In this case, a service provider is selected from a service class, defined as

a group of service providers that can deliver the same functionality. By introducing the concept of offer, our model is also applicable to the case that a provider delivers a service on more than one quality levels. An offer encapsulates both functional and non-functional aspects of a service delivery as well as the identification information of the related service provider. By this means, the service selection problem is transformed into the selection of appropriate offers: a provider is selected if and only if one of its offers is selected by a service composition. An Offer Set is defined as $OS_i = \{o_{i,j} | 1 \leq j \leq M_i\}$, where $o_{i,j}$ represents the j^{th} offer that can execute activity t_i andM_i is the number of offers in OS_i. A concrete workflow is constructed by binding each activity t_i to an appropriate offer in the related offer set OS_i, denoted as $cwf = \{t_i \leftarrow o_{i,n_i} | 1 \leq i \leq n; 1 \leq n_i \leq M_i\}$.

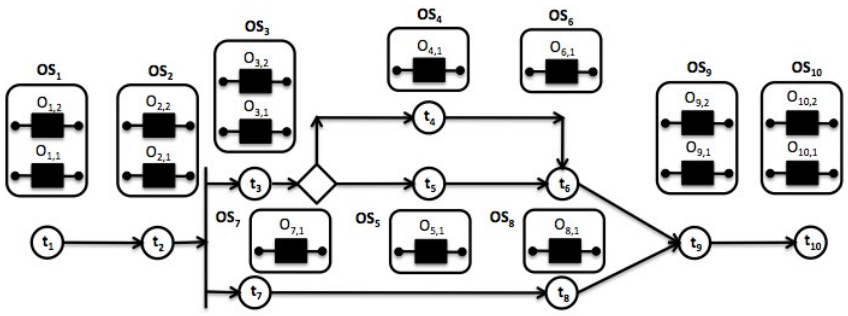


Fig. 2. Abstract Workflow

The topology of a workflow is represented by a graph where the nodes represent abstract activities and the edges specify the dependencies between these activities. In Figure 2, an abstract workflow $awf = \{t_1, t_2, \ldots, t_{10}\}$ is defined. In order to reflect different workflow patterns[1] (i.e. t_2 is followed by a parallel split while t_3 leads to conditional branches), the following two concepts are defined: an Execution Plan (EPL) epl_p is a possible execution of workflow which contains all the parallel branches but only one of conditional branches; and an Execution Path (EPA) epa_q is a sequential execution of tasks from the first activity to the last one. Let P and Q indicate respectively the number of execution plans and paths in a workflow, npl_p and npa_q indicate respectively the number of the activities included in the execution plan epl_p and execution path epa_q. The workflow shown in Figure 2 has two possible EPLs (P=2): $epl_1 = \{t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9, t_{10}\}$ and $epl_2 = \{t_1, t_2, t_3, t_5, t_6, t_7, t_8, t_9, t_{10}\}$, and three EPAs (Q=3): $epa_1 = \{t_1, t_2, t_3, t_4, t_6, t_9, t_{10}\}$, $epa_2 = \{t_1, t_2, t_3, t_5, t_6, t_9, t_{10}\}$ and $epa_3 = \{t_1, t_2, t_7, t_8, t_9, t_{10}\}$. $prob_p$ is the probability to execute the execution plan epl_p, we have $\sum_{p=1}^{p=P} prob_p = 1$.

[1] As introduced later, any workflow can be simplified and transformed into a structured Directed Acyclic Graph (DAG), thus we do not consider loops here.

2.3 Quality of Service Model

QoS Attributes The middleware system supports a large number of QoS attributes, as the limit of space, only five of them are discussed in this paper. Each offer $o_{i,j}$ specifies the value for the following QoS attributes.

- **Price**. The price $q_{pr}(o_{i,j})$ is the fee that a service requester has to pay in order to buy the offer $o_{i,j}$. Generally speaking, it can be any positive float numbers, and it is measured in dollars.
- **Time**. The execution time $q_t(o_{i,j})$ reflects the expected duration for delivering offer $o_{i,j}$. It is a positive float number measured in seconds.
- **Availability**. The availability $q_{av}(o_{i,j})$ is a real number from 0 to 1 that reflects the probability that the offer $o_{i,j}$ is deliverable.
- **Security**. The security $q_{sec}(o_{i,j})$ reflects the ability to provide authentication, authorization, confidentiality and data encryption. It has a set of enumerated values: {HIGH, MEDIUM, LOW}.
- **Reputation**. The reputation $q_{rep}(o_{i,j})$ is used to measure the trustworthiness of an offer $o_{i,j}$. Its value is calculated based on the user's feedback on his experience of purchasing $o_{i,j}$. Its value is an integer number from 1 to 5.

A QoS attribute can be either numeric or descriptive. The quality of a numeric attribute can be expressed by a number while the descriptive attribute uses an expression to describe the quality levels. Considering the five QoS attributes mentioned above, price, time, availability and reputation are numeric QoS attributes whereas security is a descriptive attribute. Moreover, a numeric QoS attribute can be either positive or negative. For positive attributes, the higher value results in a higher quality, such as availability and reputation; by contrast, take price and execution time for example, the greater value of negative attribute leads a lower quality.

Global QoS As introduced, each execution plan reflects a possible execution of a service composition. Any two individual executions can result in different aggregated QoS values (i.e. total price and execution time), depending on which execution plan is executed. The global QoS ($gqos$) defines a quality level of a service composition, which generally reflects all execution possibilities. To calculate the global QoS, the first step is to compute the aggregated QoS ($aqos$) of each execution plan. As shown in Table 1, different aggregation functions are defined for different QoS attributes.

Suppose that the provider advertises a quality level by declaring the global QoS as $gqos = (gqos_{pr}, gqos_t, gqos_{av}, gqos_{sec}, gqos_{rep})$. The global price and time are determined by the worst execution case: the aggregated price and execution time cannot exceed these limits for all execution plans. Similarly, the global security of a workflow only depends on the offers that implements the lowest security level. By contrast, the availability and reputation is calculated based on the historical information over a long periods. The values of such attributes generally reflects all execution possibilities.

Table 1. Aggregated QoS

QoS	QoS Aggregation Function	Global QoS Calculation
q_{pr}	$aqos_{pr}(epl_p) = \sum_{t_i \in epl_p, t_i \leftarrow o_{i,j}} (q_{pr}(o_{i,j}))$	$gqos_{pr} = max\{aqos_{pr}(epl_p), 1 \leq p \leq P\}$
q_t	$aqos_t(epa_q) = \sum_{t_i \in epa_q, t_i \leftarrow o_{i,j}} (q_t(o_{i,j}))$	$gqos_t = max\{aqos_t(epa_q), 1 \leq q \leq Q\}$
q_{av}	$aqos_{av}(epl_p) = \prod_{t_i \in epl_p, t_i \leftarrow o_{i,j}} (q_{av}(o_{i,j}))$	$gqos_{av} = \sum_{p=1}^{P} prob_p \cdot aqos_{av}(epl_p)$
q_{sec}	$aqos_{sec}(epl_p) = min\{q_{sec(o_{i,j})} \| t_i \in epl_p, t_i \leftarrow o_{i,j}\}$	$gqos_{sec} = min\{q_{sec}(o_{i,j}) \| t_i \leftarrow o_{i,j}\}$
q_{rep}	$aqos_{rep}(epl_p) = \frac{1}{npl_p} \cdot \sum_{t_i \in epl_p, t_i \leftarrow o_{i,j}} (q_{rep}(o_{i,j}))$	$gqos_{rep} = \sum_{p=1}^{P} prob_p \cdot aqos_{rep}(epl_p)$

3 Workflow Instantiation

The Instantiation component is defined to construct a concrete workflow *cwf* by assigning each activity t_i of an AWF to an appropriate offer $o_{i,j}$. Each instantiation request is associated with user's global QoS requirements, such as global QoS constraints $gc = (gc_{pr}, gc_t, gc_{av}, gc_{sec}, gc_{rep})$ and preferences $prefer = (p_{pr}, p_t, p_{av}, p_{sec}, p_{rep})$. The service selection is from a global perspective on the composition level ensuring that 1) the global QoS of the *cwf* can meet the requester's global constraints gc and 2) the user's preferences are maximized.

3.1 Framework of Workflow Instantiation Module

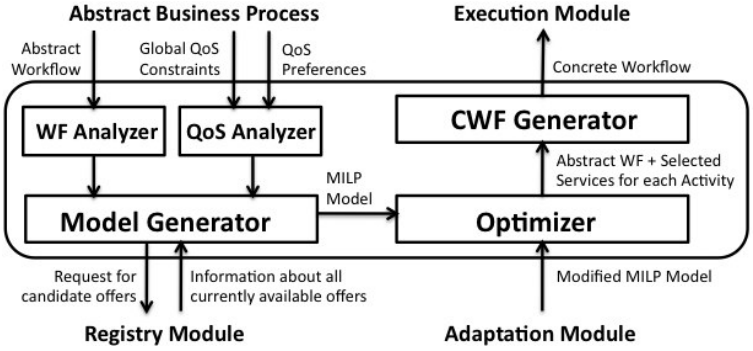


Fig. 3. Instantiation Module

The framework of the Instantiation component is shown in Figure 3. Service selection is modeled as an optimization problem. The optimization model is built based on the workflow definition as well as requester's global QoS requirements.

This information provided by the requester varies a lot for different instantiation requests, and sometimes it can be incomplete (i.e. the requester specifies global constraints only on a part of QoS attributes) or inaccuracy (i.e. his preferences are expressed in a qualitative way). WF Analyzer and QoS Analyzer modules are defined to analyze the QoS information and then generate a set of inputs to Model Generator for dynamical creation of an optimization model with respect to various requests. The service selection is modeled as a Mixed Integer Linear Programming (MILP) problem. The solution set is built by inquiring the Registry for the information about all currently available offers. On the other side, The instantiation request can also come from the Adaptation component. In this case, as detailed in Section 4.3, the Adaptation component modifies the MILP model based on the execution state of a service composition. The Optimizer solves the MILP model to select the best set of offers. The selection result is sent to CWS Generator to build a concrete workflow. Finally, the concrete workflow is forwarded to the Execution component for the execution.

3.2 Workflow Analyzer

The Workflow Analyzer module simplifies a workflow by removing circles and transforms it to a Directed Acyclic Graph (DAG). Two approaches are proposed in literature: [2] introduces an "unfolding" method: all the activities between the beginning and the end of a cycle are cloned K times, where K is the maximum number of times that each cycle is executed according to the past execution logs. In [3], loops peeling is introduced as an improvement of unfolding technique: loop iterations are represented as a sequence of branches. Each branch condition evaluates if the loop l has to continue or exit.

The objective of workflow analysis is to generate two sets of vectors expressing the workflow structure. Considering a workflow with N activities, for each EPL $epl_p(1 \leq p \leq P)$, a N-dimensional vector $isEpl_p = (u_{p,1}, ..., u_{p,N})$ is created as follows: if activity t_i is included in epl_p, $u_{p,i}$ is set to 1; otherwise, it is set to 0. For example, after analyzing the workflow defined in Figure 2, the vector $isEpl_1 = (1, 1, 1, 1, 0, 1, 1, 1, 1, 1)$ is generated for epl_1. Likewise, a set of vectors $isEpa_q = (v_{q,1}, ..., v_{q,N})(1 \leq q \leq Q)$ are created for each EPA epa_q.

3.3 QoS Analyzer

The optimization models proposed in most of literatures are static. They assume that the requester specifies global constraints and preferences on all QoS dimensions. However, in practical cases, the global QoS requirements provided by clients can be incomplete or inaccurate. QoS Analyzer is defined to deal with such information and generate a set of input for Model Generator. In order to set up a mathematical model, all descriptive QoS attributes need to be quantified first. QoS Analyzer assigns a suitable integers to each enumerated value, based on the semantic. Within the scope of this paper, only security is a descriptive attribute, and its enumerated values are assigned respectively as follows: LOW=0, MEDIUM=1 and HIGH=2. Hereafter, security is regarded as

a numeric attribute. Furthermore, the system supports a large number of QoS attributes to meet the different QoS requirements of various clients. But in practice, each client always specifies global constraints on several of them. In this case, a 5-dimensional binary vector $Z = (z_1, ..., z_5)$, is created as follows: z_r ($1 \leq r \leq 5$) equals to 1 if the user has specified the global constraint on the r^{th} QoS dimension; otherwise, it is set to 0. Hereafter, to facilitate the mathematical expressions, the QoS attributes are indexed as follows: price=1, time=2, availability=3, security=4 and reputation=5. For example, if a client has limited budget and delivery time and the global QoS constraints are expressed as $gc = \{\$10, 200s\}$. In this case, $Z = (1, 1, 0, 0, 0)$.

In most of practical cases, it is hard for a client to give a complete preference on all QoS attributes; instead, his interests only focus on several of them. As an example, the client only specifies his preferences on security issue and reputation as $prefer = (-, -, -, 2, 3)$. In this case, QoS Analyzer completes the preference vector by assigning 0 to all the QoS attributes that are not mentioned, and then normalizes the preference vector. In this example, the client's preference is finally expressed as $prefer = (0, 0, 0, 0.4, 0.6)$. However, as discussed in [4], requesters can hardly express preferences in a quantitative way. Instead, he can only tell which attribute is relatively more important for him. In this case, the requester's preferences are expressed by a set of qualitative expressions such as $q_t > q_{pr}$. QoS Analyzer quantizes and completes the preference vector by taking the following steps: 1) set the preferences on all the QoS dimensions that are not mentioned by the client to 0; 2) for each preference expression $q_L > q_R$, the preference on q_L is set as twice as the one on q_R; 3) If two preferences are not comparable, give them the same weight; 4) finally, all attributes can be expressed using one attribute; 5) calculate the preferences on this attribute by applying $\sum_{k=1}^{k=5} p_k = 1$ and 6) the preferences on all attributes can be calculated. For example, if the preferences are expressed as follows: *i*) price>time; *ii*) security>time, finally it can be quantified as $prefer = \{0, 4, 0.2, 0, 0.4, 0\}$.

3.4 Integer Programming Model

Objective Function. As adopted in [2], [1], [3], [5], Simple Additive Weighting (SAW) technique is used to set up the objective function in order to determine the best service composition. There are two phases to apply SAW:

- Scaling phase. Different QoS attributes have different value ranges and their values are not comparable. On the other hand, the values of positive attributes have to be maximized while the values of negative attributes have to be minimized. As a result, in order to use an objective function to evaluate the overall QoS performances of a service composition, the negative attributes are scaled according to Formula (1) whereas the positive ones are scaled according to Formula (2).

$$\text{Negative } :nqos_k = \begin{cases} \frac{gqos_k^{max} - gqos_k}{gqos_k^{max} - gqos_k^{min}} & if\ gqos_k^{max} \neq gqos_k^{min} \\ 1 & otherwise \end{cases} \quad (1)$$

$$\text{Positive}: nqos_k = \begin{cases} \frac{gqos_k - gqos_k^{min}}{gqos_k^{max} - gqos_k^{min}} & if\ gqos_k^{max} \neq gqos_k^{min} \\ 1 & otherwise \end{cases} \quad (2)$$

$gqos_k^{max}/gqos_k^{min}$ is the maximum/minimum value of $gqos_k$. To calculate $gqos_k^{max}/gqos_k^{min}$, it is not necessary to generate all possible concrete workflows. For each offer set, the offer with the greatest/smallest value on the k^{th} QoS dimension is selected. The computation complexity is proved as polynomial in [2].

– Weighting phase. The overall score of a service composition is calculated based on the normalized QoS values and the user's preferences on different attributes. The objective function of MILP model is to maximize this score, given by the Formula (3).

$$\text{Objective}: \max \sum_{k=1}^{5} prefer_k \cdot nqos_k \quad (3)$$

Variables. A binary variable $x_{i,j}$ is created for each offer $o_{i,j}$ indicating whether it is selected: if $o_{i,j}$ is selected, $x_{i,j}$ is set to 1; otherwise set to 0.

Constraints. First of all, exact one offer in an offer set OS_i can be selected:

$$C_0 : \sum_{j=1}^{M_i} x_{i,j} = 1; \quad \forall i, 1 \leq i \leq N \quad (4)$$

In the following, constraints C_1 to C_5 are defined to guarantee the global QoS constraints. C_1 enables that the global constraint on price can be fulfilled:

$$C_1 : \sum_{i=1}^{N} \sum_{j=1}^{M_i} isEpl_{p,i} \cdot x_{i,j} \cdot q_{pr}(o_{i,j}) \leq gc_{pr}; \quad \forall p, 1 \leq p \leq P \quad (5)$$

According to Table 1, the global price is determined by the most expensive execution plan. The global constraint gc_{pr} is satisfied when the aggregated price $aqos_{pr}$ of each execution plan is under this limit. Here, the information provided by WF Analyzer ($isEpl_{p,i}$) is used to determine whether an activity is included in an execution plan epl_p. The global constraint on time is denoted in the similar way by C_2. According to Table 1, the aggregated execution time is calculated based on execution path.

$$C_2 : \sum_{i=1}^{N} \sum_{j=1}^{M_i} isEpa_{p,i} \cdot x_{i,j} \cdot q_t(o_{i,j}) \leq gc_t; \quad \forall q, 1 \leq q \leq Q \quad (6)$$

The aggregated availability of each execution plan epl_p is calculated as follows:

$$aqos^p_{av} = \prod_{i=1}^{N} \sum_{j=1}^{M_i} isEpl_{p,i} \cdot x_{i,j} \cdot q_{av}(o_{i,j}); \quad \forall p, 1 \leq p \leq P \tag{7}$$

Based on Formulas (4), Formulas (7) can be linearized as follows:

$$lq^p_{av} = \sum_{i=1}^{N} \sum_{j=1}^{M_i} isEpl_{p,i} \cdot x_{i,j} \cdot ln(q_{av}(_{i,j})); \quad \forall p, 1 \leq p \leq P \tag{8}$$

where $lq^p_{av} = ln(aqos^p_{av})$. According to Table 1, the global constraint on the availability is expressed as:

$$C_3 : \sum_{p=1}^{P} e^{lq^p_{av}} \cdot prob_p \geq gc_{av}; \tag{9}$$

The global security depends on the Web service that implements the lowest security level, so the global constraints on security level is denoted as follows:

$$C_4 : \sum_{j=1}^{M_i} x_{i,j} \cdot q_{sec}(o_{i,j}) \geq gc_{sec}; \quad \forall i, 1 \leq i \leq N \tag{10}$$

Finally, the aggregated reputation for each execution plan epl_p is calculated according to Formulas (11):

$$aqos^p_{rep} = \frac{1}{npl_p} \cdot \sum_{i=1}^{N} \sum_{j=1}^{M_i} isEpl_{p,i} \cdot x_{i,j} \cdot q_{rep}(o_{i,j}); \quad \forall p, 1 \leq p \leq P \tag{11}$$

And the global constraints on reputation is denoted by C_5:

$$C_5 : \sum_{p=1}^{P} aqos^p_{rep} \cdot prob_p \geq gc_{rep}; \tag{12}$$

As we argued, most of literature assume that requester specifies global constraints on all QoS attributes. However, the global constraints are usually incomplete in practice. In our middleware system, Model Generator module dynamically generates constraints for the attributes with global constraints. The constraint C_i $(1 \leq i \leq 5)$ is added to the MILP model if and only if z_i equals to 1. Remember that the QoS Analyzer generates a vector $Z = (z_1 : z_5)$ indicating whether the user limits global constraint on each QoS attribute.

4 Workflow Execution, Monitoring and Adaptation

4.1 Execution of Workflow

Most of traditional service middlewares/platforms implement a centralized execution engine to conduct and control the executions of business processes. As

discussed in [6], centralized execution results in a series of drawbacks in dealing with scalability, fault tolerance and privacy. In our middleware system, the execution of a business process is decentralized. After the instantiation, a concrete workflow is constructed and sent to the Execution component for executing. Each execution request is associated with the invocation parameters as well as a unique ID number, which is provided by the ABP before the instantiation. Based on the concrete workflow, the Execution component generates a set of coordination messages and then forwards them to all partner services. A coordination message describes the workflow execution from a local perspective of each participant, such as its precedent(s), successor(s), the local QoS expectations as well as the execution ID.

The workflow is executed in a collaborative manner: upon receiving the coordination message, each partner service involved in the workflow knows the part it plays in the interactions so that it only interacts with its "neighbors" rather than a specific business process that a single engine executes. After the distribution of the coordination messages, the Execution component packages the execution ID number together with invocation parameters into an invocation message, and then forwards it to the AS that is bound to the first activity of the workflow. For each partner service, once all inputs with the expected execution ID are received, it starts the execution and then encapsulates the result and execution ID into a new invocation message, which is then sent to its successor(s). By this means, the workflow is executed as a dominos falling started from the first activity, and each participant evolved in a service composition executes its part according to the behaviors of other participants, there is no single point of control.

4.2 Monitor the Execution of Workflow

During the execution of workflow, run-time QoS of partner services cannot be always ensured due to various aspects, such as the network congestion or the increasing load on servers. Thus, in order to ensure requester's global constraints on each execution, the `Monitor` component is designed to monitor the run-time QoS of all partner services. If a global QoS violation is predicted, the adaptation decision is made in order to re-select a set of services with better QoS characteristics for the unexecuted activities. Among the five QoS attributes discussed in this paper, we assume that service providers do not change the price and security policies during the execution. The reputation depends on the preferences of requesters and its value calculated based on historical information over a relatively long period, we suppose its value changes slightly during the period of an execution. As a result, we only discuss execution time and availability at run-time.

It is not extensively studied in the related literature on how to monitor the executions of service compositions and draw the appropriate decisions to recompose the rest of workflow. In [2] [1] [3], the authors list a number of events that can trigger the service re-composition. For example, the authors state in [3] that the re-composition is triggered if the current value of q_i differs from the expected value q_i' by more than a given threshold Δ_i, i.e., $|q_i - q_i'| > \Delta_i$. But how to

evaluate the global QoS q_i at run-time is not discussed in [3]. In the following, we introduce 2 monitoring techniques: Program Evaluation and Review Technique (PERT) is used to evaluate the global execution time, and Heartbeat Failure Detector ($\mathcal{HB}$) is adopted to estimate the availability of partner services.

PERT Chart. PERT is developed as a well-known approach for planning, monitoring and managing the progress of complex projects [17]. Based on the concrete workflow to be monitored, the `Monitoring` component builds up a variation of PERT chart by calculating the following information for each activity:

- The earliest start time T_E. This is the earliest time an activity t_i can start in case that all its precedents accomplish the task on schedule. $T_E(t_i) = max\{T_E(t_j)+q_t(t_j)|t_j \in precedent(t_i)\}$. For the first activity t_1, $T_E(t_1) = 0$.
- The latest start time T_L. This is the latest time by which an activity t_i can begin without causing the delay of the whole execution of workflow. $T_L(t_i) = min\{T_L(t_k) - q_t(t_i)|t_k \in successor(t_i)\}$. For the last activity t_N, $T_E(t_N) = gqos_t - q_t(t_N)$.
- The critical path CP. The critical path is identified by the activities that has no slack time, denoted as $CP = \{t_i|T_E(t_i) = T_L(t_i)\}$.
- The global slack time S_G. The maximum delay that an execution of workflow can produce without causing the violation of global time constraint. $S_G = gc_t - gqos_t$.

Similar to the distribution of workflow execution, the PERT chart is also distributed so that each partner service has local knowledge of the global execution time. By this means, the execution time is monitored by all partner services in a decentralized manner. The Monitoring component re-calculates the PERT chart when a negative event (such as a delay) is produced. Afterwards, it updates the local view of (a part of) partner services by diffusing a set of update messages.

We assume that the system has a synchronized clock and no clock drift. In this context, each partner service bound to t_i read its local time. If the expected invocation message is not received by $T_L(t_i)$, it reports the delay to the Monitoring component. And then, it extends $T_L(t_i)$ by $S_G+\Delta_t$. During this period, if no invocation comes yet, it declares the possibility of the global time violation to the Monitoring component. Upon receiving the first alert message, the Monitoring component prepares the adaptation, such as saving the breakpoints and update the global view of the execution state. When the second message arrives, it draws the adaptation decision and forwards all the information related to this execution to the Adaptation component. The margin Δ_t is calculated with respect to $gqos_t$ and S_G. It defines a reasonable tolerant range to prevent an early decision that may lead an unnecessary adaptation process.

Using the PERT technique, it is easy to decide whether to adapt the rest of workflow when a delay is produced during the execution of an activity. But it cannot efficiently cope with the case that a partner service is crashed. For example, the activity t_i starts to execute at T, and the service bound to t_i is crashed at $T + \Delta$, where $\Delta \approx 0$. In this case, the workflow execution is

blocked until an instant that one of its successors, noted as t_j, reports global time violation since $T_L(t_j)$ is reached. In this case, t_i has to be re-executed and the delay is $T_L(t_j)$-$T_E(t_i)$+S_G+Δ_t. To cope with this problem by making decision earlier, an overlay is built based on failure detector .

Heartbeat Failure Detector. Failure detector is proposed as a basic building block for fault-tolerant distributed systems in 1990s [7]. Roughly speaking, it is a sub-system responsible for detecting node failures or crashes in a distributed system. In this paper, we investigate *heartbeat* failure detector ($\mathcal{HB}$) proposed in [8]. The basic idea of $\mathcal{HB}$ is that each component diffuses a $\mathcal{HB}$ message "I am alive" periodically to all the other components in the system. On the other hand, a component is suspected as crashed by others if its $\mathcal{HB}$ is lost. In order to draw a quick adaptation decision when a partner service becomes unavailable, a $\mathcal{HB}$ overlay is built on top of all ASs based on the algorithm defined in Figure 4.

1: Each partner service ps_i:
2: *Initialization*:
3: $T_i^0 \leftarrow \max\{0, T_E(t_i) - \Delta_s\}$;
4: **if** $t_i \in CP$: Δ=Δ_{CP}; **else**: $\Delta = \Delta_{NC}$;
5: *Task 1*: send a $\mathcal{HB}$ message to *cs*
6: at time $T_i^k = T_i^0 + k \cdot \Delta$, send the k^{th} $\mathcal{HB}$ message m_i^k to *cs*;
7: The service composition *cs*:
8: *Instantiation*:
9: *suspected*$\leftarrow \emptyset$; $\forall ps_i$: $expected_i$=0;
10: *Task 2*: receive the k^{th} $\mathcal{HB}$ message m_i^k from ps_i
11: **if** $k = expected_i$:
12: **if** $ps_i \in$ *suspected*: *suspected*$\leftarrow$ *suspected*$\backslash\{ps_i\}$
13: *Task 2.1*: estimate the next freshness point τ_i^{k+1}
14: $T_i^k \leftarrow$ current time; $expected_i \leftarrow expected_i + 1$;
15: **if** k $< L$: $EA_i^{k+1} = (\text{k+1})\cdot\Delta + C_0$;
16: **else**: $EA_i^{k+1} = (\text{k+1})\cdot\Delta + \frac{1}{L}\cdot \sum_{j=k-L}^{j=k}(T_i^j \text{ - } \Delta\cdot \text{ i})$;
17: $\tau_i^{k+1} = EA_i^{k+1} + \alpha_0$;
18: **else if** $k = expected_i$-1:
19: $T_i^k \leftarrow$current time; *suspected*$\leftarrow$ *suspected*$\backslash\{ps_i\}$
20: *Task 3*: up to τ_i^k, m_i^k is not received yet
21: **if** $ps_i \notin$ *suspected*:
22: *suspected*$\leftarrow$ *suspected* $\cup \{ps_i\}$; **do** *Task 2.1*;
23: **else**: predict the crash of ps_i and decide adaptation.

Fig. 4. The implementation of $\mathcal{HB}$ based overlay

First of all, each partner service ps_i initializes the starting time T_i^0 by which it is required to send $\mathcal{HB}$ messages. The time interval between 2 consecutive $\mathcal{HB}$ messages Δ is set to Δ_{CP} if ps_i is on the critical path and set to Δ_{NC}

otherwise (line 4). For the service composition cs, when the $\mathcal{HB}$ message with the expected sequence number is received from ps_i (line 11), if ps_i is suspected, it is removed from the suspected list (line 12). And then, it records the receive time and update the expected sequence number. In the following, it calculates the next freshness point based on the estimation of the arrival time of the next $\mathcal{HB}$ and a constant safety margin α_0 (line 17). A freshness point τ_i^k is the firm deadline for the k^{th} $\mathcal{HB}$ since cs starts to suspect ps_i if the expected $\mathcal{HB}$ is not received. The estimation of the arrival time for the next $\mathcal{HB}$ is based on the approach proposed in [9] which considers the L most recent $\mathcal{HB}$s (line 16). A static estimation approach is applied for the first k $\mathcal{HB}$s when $k < L$ (line 15). *Task 3* is executed if the expected $\mathcal{HB}$ is not received by its freshness point. If ps_i is not suspected, it is added into the suspected list. In order to avoid a hurried decision, the adaptation decision is not made once ps_i is suspected. Instead, we assume that it was received and estimate the next freshness point by carrying out *Task 2.1* (line 22). By τ_i^{k+1}, if the k^{th} $\mathcal{HB}$ message is received, ps_i is removed from the suspected list (line 19). Finally, an adaptation decision is driven by two consecutive failures on receiving a $\mathcal{HB}$ messages (line 23).

As discussed in [9], the QoS of a failure detector is evaluated from two aspects: the detection time (T_D) measures how fast it can detect a failure and mistake rate (MR) reflects the percentage of fault decisions. In our algorithm, the next freshness point is determined by Δ, α_0 and average delay (line 16-17). Considering the average delay is negligible compare to other parameters, the detection time T_D depends on both Δ and α_0. Once a partner service ps_i is crashed, its failure is detected after two freshness points so that $T_D \approx 2\Delta + 2\alpha$. In [10], authors evaluates this estimation approach adopted in our algorithm. It is reported that the average mistake rate MR is around 0.1%. In our algorithm, an adaptation decision is made on when 2 consecutive fault suspects are produced. Thus, the MR is negligible in the real system.

4.3 Workflow Adaptation

Once `Monitoring` component predicts global QoS violation with current composition *cwf*, it forwards all the information related to this execution to the `Adaptation` component. Such information includes workflow definition, binding information as well as the expected QoS and execution state of each activity. Based on the execution state, the activities of the workflow are divided into 2 parts: $A_F = \{t_i | \text{the execution of } t_i \text{ is finished}\}$ and $A_N = \{t_j | t_j \notin A_F\}$. The adaptation attempts to identify a new service composition cwf' that can still meet the global QoS constraints specified by the requester. cwf' is identified by re-selecting and re-binding a set of services with better QoS characteristics for the activities in A_N.

Accordingly, the adaptation can be seen as the re-instantiation of a part of workflow. The `Adaptation` component creates a new MILP model by modifying the former one. The modification is based on the execution state and all currently available offers. Since each activity t_i in A_F is already executed, the selection variables $x_{i,j}$ $(1 \leq j \leq M_i)$ of all offers in the offer set OS_i have to

be fixed as new constraints. On the other hand, for A_N, the offer sets are updated by inquiring the `Registry` component for all currently available offers. In this way, the selection is only performed for A_N. Apart from the new MILP model, `Adaptation` component also decides an adaptation strategy for run-time re-instantiation process: either a feasible or the optimal solution is identified. The decision is made based on the information of the first instantiation process encapsulated in the *cwf*, such as the time costs and objective values of both a feasible solution and the optimal one. Such information can be used to predict the up-coming service selection process, i.e., as argued later by the experimental results, a feasible strategy is decided under a severe QoS constraints.

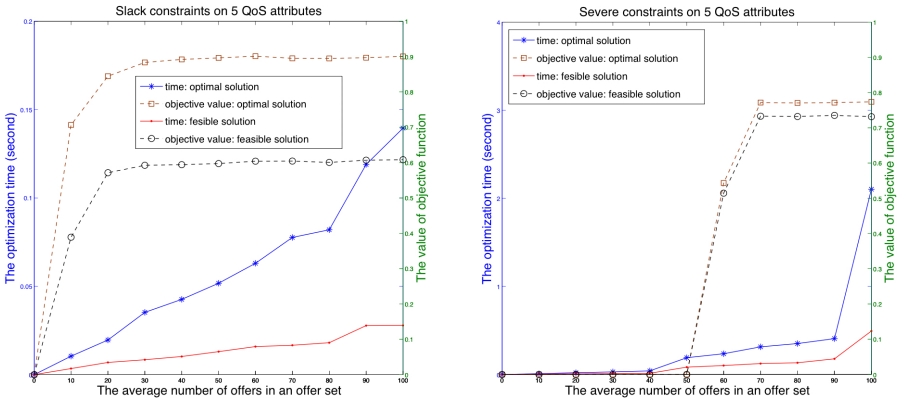
The `Instantiation` component solves the new MILP model under the specified strategy and finally constructs a new service composition cwf'. All information related to both *cwf* and cwf' is then forwarded to `Execution` component. Before resuming the execution, it compares both *cwf* and cwf' and updates the local perspectives of all partner services. New coordination messages are created for the partner services that are involved in cwf' but are not in *cwf*. If a service provider is not longer selected in cwf', an invalidation message is create to cancel its participation in the execution of a service composition. The changes of binding information result in updating the knowledge of neighbors for some partner services. After the distribution of such updated coordination messages, the execution is resumed from the breakpoints.

5 Experiment Results

Experiment Set Up. The performance of run-time service selection depends on the execution cases, i.e., the size and complexity of a workflow, the average size of offer sets, the global QoS constraints as well as the optimization strategy. First of all, we investigate the instantiation of simple workflow: two instantiation scenarios are set up based on the workflow defined in Figure 2 with respectively slack and severe global QoS constraints. For each activity t_i, a complete offer set OS_i is created by randomly generating more than 100 offers. Each offer $o_{i,j}$ is defined by initializing the values of different QoS attributes as follows: the execution time q_t is generated randomly assuming a uniform distribution in the interval 10 and 200. The security level q_{sec} is randomly chosen from the enumerated values with the same probability. The availability is determined by creating a Gaussian random number between 0 and 1 with μ=0,98 and σ=0.02. The reputation is generated in the similar way by generating a Gaussian random number r between 0 and 5 with μ=4 and σ=1. As the reputation is integer value, q_{rep} is then set to $\lceil r \rceil$. Finally, the price is calculated with respect to the other 4 attributes.

For each scenario, a group of testing cases are generated by varying the size of offer sets (M_i) for each activity from 1 to 100. For each case with $M_i = N(1 \leq N \leq 100)$, the workflow is instantiated for 5 times. Each instantiation identifies a feasible solution as well as the optimal one based on randomly selected offer sets. The average time cost and objective value of both strategies are calculated

for each testing case. The experiment results shown in Figure 5 is grouped by 10 consecutive testing cases: each point represents the average performance of 10 test cases (i.e. the objective value when N=70 is the mean value of the testing cases with N respectively from 61 to 70). Figure 5(a) reflects instantiation scenario with slack global QoS constraints. The time to identify the optimal and a feasible solution is respectively within 150ms and 100ms. In the real system, this difference is negligible. On the other side, the objective value of the optimal solution is about 30% higher than a feasible solution. Figure 5(b) illustrates the experimental results for instantiation requests associated with a severe global QoS constraints. While the number of offers per activity is less then 50, no possible concrete workflow can be identified. As the size of offer sets grows large (with from 90 to 100 offers), the optimization time increases dramatically by nearly 5 times. By contrast, the time cost for a feasible solution increases slowly and always within 500ms. On the other side, the values of objective function for both strategies differ by only 5%.



(a) Slack Global QoS Constraints (b) Severe Global QoS Constraints

Fig. 5. Experiment Results

Moreover, another group of experiments are made in the same way based on a more complex workflow which composes more than 50 activities and dozens of execution paths and plans. The first instantiation scenario is based on medium global QoS constraints: the optimization time raises to 1 minute for identifying the optimal solution. But a feasible solution can be always identified within 1 second. The objective values differ by 10%-15%. And then, the experiment is re-conducted under severe global constraints which leads the difference of objective values around 5%. In this case, the time complexity of the optimal solution does not change a lot and a feasible solution can also be identified within 5 seconds. Considering the longer execution time for a more complex workflow, a feasible solution can meet the crucial requirement of limited execution time. From the

aforementioned experimental results, we have the following conclusion: 1) the optimal service composition can be identified efficiently for the workflow with a rational size (10-15 activities) and complexity (less than 5 splits); 2) as the time complexity is exponential with respect to the complexity of workflow whereas the execution time grows linearly, a feasible solution is more efficient for the complex instantiation scenarios; 3) when the global QoS is severe, the values of objective function for a feasible solution and the optimal one differs only around 5%; In this case, a feasible solution can identified instead of the optimal one for at run-time since it has almost the same quality level but can be identified much faster.

6 Related Work

Dynamic service composition has attracted great interests in the research community. In [2], the authors present AgFlow: a middleware platform that enables QoS driven service composition. Two alternative service selection approaches are discussed: local optimization and global planning. The local approach implements a greedy algorithm that selects the best candidate for each activity individually; accordingly the end-to-end QoS is not guaranteed. The global approach selects the services by separately optimizing each execution path. As demonstrated in [1], this global approach cannot always guarantee the global QoS. In [11], the authors presented a hybrid approach that combines global optimization with local selection. The idea is to firstly decompose global QoS constraints into a set of local ones by using Mixed Integer Programing (MIP), and then the best candidate is selected locally for each activity independently. When QoS decomposition results in a set of restrictive local constraints, the failure of local selection causes the failure of identifying a service composition, although a solution may actually exist.

In order to select services from the global perspective on service composition level and ensure the end-to-end QoS, the authors in [12] propose two models: combinatorial model and the graph model. The service selection is defined as a Multiple-choice Multiple-dimension Knapsack Problem (MMKP) in the former model and defined as a Multi-Constraint Optimal Path (MCOP) problem in the latter one. But the combinational model is only suitable for linear workflows. In [1], the service selection is modeled as a Mixed Integer Linear Programming (MILP) problem where both local and global QoS constraints can be specified. The fulfillment of global QoS is guaranteed. Their work is later extended in [3] by introducing loop peeling and negotiation technique. The experimental results shows that it is effective for large processes even with severe constraints. As argued in [13], linear programming method is not suitable for run-time service selection since the time complexity to obtain the optimal solution is exponential. In addition, other approaches proposes heuristic algorithms to efficiently find a near-optimal solution. In [14], the authors extend both models they proposed in [12] to the general workflow case and introduce a heuristic algorithm for each model to find near-optimal solutions in polynomial time, which is more suitable

for real-time service selections. In [5], authors present a heuristic algorithm based on clustering techniques, which shows a satisfying efficiency in terms of time cost and optimality.

However, most of aforementioned approaches are based on quantitative criteria to select optimal/near-optimal solutions, such as utilizing an objective function. As discussed in [4], it is difficult for users to express their preferences using quantitative metrics. The authors then introduce CP-nets for conducting qualitative Web service selection. Since qualitative preferences can be incomplete, the selection may result in a set of incomparable candidates. Their work is improved later in [15] by using historical information to complete preferences in order to reduce the size of result set. But this work is limited to local service selection.

In our previous work [16], a middleware architecture is introduced for dynamic service selection and execution: partner services are selected and bound at run-time and the aggregated QoS of a service composition is ensured to meet the requester's end-to-end QoS requirement. This paper improves and completes the previous work by introducing adaptive service execution. The adaptive service execution is not extensively studied in the related literature. In [2], [3], authors list a number of events that can trigger run-time service recomposition. But all these events are described from a local perspective without the consideration of the workflow structures and the execution state. In addition, it is inefficient to deal with the case that partner services can crash permanently during the execution. We promote the aforementioned work by introducing a comprehensive middleware system that integrates service selection, execution, monitoring and adaptation to meet the requirement of dynamic and adaptive service execution. By introducing PERT and $\mathcal{HB}$ failure detector as two effective monitoring approaches, the system is able to react to the dynamic execution environment and draw appropriate adaptation decisions.

7 Conclusion

In this paper, we present a QoS-aware middleware system for dynamic and adaptive service execution. Compare to the work reported in the literature, our system has the following desirable characteristics: 1) The flexible generation of MILP model can deal with incomplete and inaccurate QoS requirements; 2) execution of a service composition is decentralized; 3) PERT and $\mathcal{HB}$ failure detector are explored as two efficient monitoring approaches, which can deal with both delay and crash; 4) based on the different execution requests and run-time execution states, a suitable strategy is decided for real-time adaptation. In the future, this work is extended by setting up an adaptive and distributed monitoring overlay. We are studying on a distributed algorithm by integrating both PERT and $\mathcal{HB}$ approaches for monitoring decentralized service execution. Moreover, the monitoring overlay is adaptable to different execution requests: based on the different execution scenarios (workflow definitions, QoS requirements etc.) the monitoring of a service execution are provided on various quality levels by dynamically configuring the parameters of the monitoring overlay.

References

1. Ardagna, D., Pernici, B.: Global and local qos constraints guarantee in web service selection. In: Proceedings of the IEEE International Conference on Web Services. ICWS '05, Washington, DC, USA, IEEE Computer Society (2005) 805–806
2. Zeng, L., Benatallah, B., H.H. Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. IEEE Trans. Softw. Eng. **30** (May 2004) 311–327
3. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Trans. Softw. Eng. **33** (June 2007) 369–384
4. Wang, H., Xu, J., Li, P.: Incomplete preference-driven web service selection. In: Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 1, Washington, DC, USA, IEEE Computer Society (2008) 75–82
5. Mabrouk, N.B., Beauche, S., Kuznetsova, E., Georgantas, N., Issarny, V.: Qos-aware service composition in dynamic service oriented environments. In: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware. Middleware '09, New York, NY, USA (2009) 7:1–7:20
6. Fernández, H., Priol, T., Tedeschi, C.: Decentralized Approach for Execution of Composite Web Services using the Chemical Paradigm. 8th International Conference on Web Services (ICWS 2010) (2010) 139–146
7. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43** (March 1996) 225–267
8. Aguilera, M.K., Chen, W., Toueg, S.: Heartbeat: A timeout-free failure detector for quiescent reliable communication. Technical report, Cornell University, Ithaca, NY, USA (1997)
9. Chen, W., Toueg, S., Aguilera, M.K.: On the quality of service of failure detectors. IEEE Transactions on Computers **51** (2002) 13–32
10. Bertier, M., Marin, O., Sens, P.: Implementation and performance evaluation of an adaptable failure detector. In: Proceedings of the 2002 International Conference on Dependable Systems and Networks. DSN '02, Washington, DC, USA, IEEE Computer Society (2002) 354–363
11. Alrifai, M., Risse, T.: Combining global optimization with local selection for efficient qos-aware service composition. In: Proceedings of the 18th international conference on World wide web. WWW '09, New York, NY, USA, ACM (2009) 881–890
12. Yu, T., Lin, K.J.: Service selection algorithms for composing complex services with multiple qos constraints. In: ICSOC'05: 3rd Int. Conf. on Service Oriented Computing (2005) 130–143
13. Alrifai, M., Skoutas, D., Risse, T.: Selecting skyline services for qos-based web service composition. In: Proceedings of the 19th international conference on World wide web. WWW '10, New York, NY, USA, ACM (2010) 11–20
14. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end qos constraints. ACM Trans. Web **1** (May 2007)
15. Wang, H., Shao, S., Zhou, X., Wan, C., Bouguettaya, A.: Web service selection with incomplete or inconsistent user preferences. In: International Conference on Service Oriented Computing. (2009) 83–98
16. Napoli, C.D., Giordano, M., Pazat, J.L., Wang, C.: A chemical based middleware for workflow instantiation and execution. In: ServiceWave. (2010) 100–111
17. Wiest, J.D., Levy;.F.K.: A Management Guide to PERT/CPM. Prentice- Prentice Hall (1969)
18. OASIS Standard: Web Services Business Process Execution Language. (2007)